perlpod

Table des matières

1 NAME/NOM

perlpod - Le format Pod (plain old documentation), la bonne vieille documentation

2 DESCRIPTION

Pod est un langage de balise, simple à utiliser pour écrire de la documentation pour Perl lui-même ainsi que pour les programmes Perl et les modules Perl.

Des traducteurs existent pour convertir le Pod vers différents formats comme le texte brut, le HTML, les pages man et d'autres encore.

Le langage Pod reconnait à la base trois sortes de paragraphes : les paragraphes ordinaires (§2.1), les paragraphes de commande (§2.3) et les paragraphes verbatim (§2.2).

2.1 Les paragraphes ordinaires

La plupart des paragraphes d'une documentation sont des blocs de texte ordinaires, comme celui-ci. Il vous suffit de taper votre texte sans aucune marque particulière et avec une ligne vide avant et après. Lorsqu'il sera formaté, ce bloc subira une mise en forme minimale comme un redécoupage des lignes, problablement écrites dans une police à espacement proportionnelle, qui seront sans doute justifiées.

Dans les paragraphes ordinaires, vous pouvez utiliser des codes de mise en forme pour le **gras**, l'*italique*, le style `code`, les liens et autres. Ces codes sont expliqués dans la section Codes de mise en forme (§2.4), ci-dessous.

2.2 Les paragraphes verbatim

Les paragraphes verbatim (mot pout mot) sont habituellement utilisés pour présenter des blocs de code ou d'autres bouts de texte qui ne requièrent aucune analyse, aucune mise en forme particulière et dont les lignes ne doivent pas être redécoupées.

Un paragraphe verbatim se distingue par son premier caractère qui doit être un espace ou une tabulation. (Et habituellement, chacune de ses lignes commence par des espaces ou des tabulations.) Il devrait être reproduit à l'identique, en supposant que les tabulations sont alignées sur 8 caractères. Il n'existe aucun code de mise en forme et, par conséquent, aucune possibilité de faire de l'italique ou quoi que ce soit d'autre. Un \ est un \ et rien d'autre.

2.3 Les paragraphes de commande

Un paragraphe de commande est utilisé pour spécifier des traitements spéciaux sur des parties du texte comme les titres ou les listes.

Tous les paragraphes de commande (qui ne font habituellement qu'une seule ligne) commencent par le caractère « = », suivi d'un identificateur, suivi d'un texte arbitraire que la commande peut utiliser de la façon qui lui plaît. Les commandes actuellement reconnues sont

```
=pod
=head1 titre
=head2 titre
=head3 titre
=head4 titre
=over niveauindentation
=item texte
=back
=beagin format
=end format
=for format
=encoding type
=cut
```

Voici en détail des explications pour chacune d'elles :

=head1 *Texte du titre*

=head2 *Texte du titre*

=head3 *Texte du titre*

=head4 *Texte du titre*

Les commandes head1 à head4 produisent des titres et head1 est le titre de plus haut niveau. Le texte qui suit la commande et qui constitue le reste du paragraphe est le contenu du titre. Par exemple :

```
=head2 Attributs des objets
```

Le texte "Attributs des objets" est ici le titre. (Notez que les niveaux head3 et head4 sont des ajouts récents qui ne seront pas reconnus par de vieux traducteurs de Pod.) Le texte du titre peut utiliser des codes de mise en forme comme dans :

```
=head2 Valeurs possibles pour C<$/>
```

Ces codes sont expliqués dans la section Codes de mise en forme (§2.4), ci-dessous.

=over *niveauindentation*

=item *texte...*

=back

Les commandes item, over, et back ont besoin d'un peu plus d'explications : « =over » débute une section destinée à créer une liste utilisant des commandes « =item », ou pour indenter un ou plusieurs paragraphes normaux. Utilisez « =back » à la fin de votre liste ou de votre groupe de paragraphes. L'option *niveauindentation* de « =over » indique le niveau d'indentation, généralement mesuré en em (où un em est la largeur d'un M de la police de base du document) ou en une unité comparable. Si l'option *niveauindentation* est omise, sa valeur par défaut est quatre. (Et certains traducteurs ignoreront cette valeur quelle qu'elle soit.) Dans le *texte* de `=item texte...` vous pouvez utiliser des codes de mise en forme comme par exemple :

```
=item Utilisation de C<$|> pour contrôler l'usage des tampons
```

Ces codes sont expliqués dans la section Codes de mise en forme (§2.4), ci-dessous.

Notez aussi les quelques règles basiques suivantes pour bien utiliser les sections « =over » ... « =back » :

- N'utilisez pas « =item » en dehors d'une section « =over » ... « =back ».
- La première chose qui suit une commande « =over » devrait être une commande « =item », sauf s'il n'y a vraiment aucun item dans cette section « =over » ... « =back ».
- N'utilisez pas de commande « =head*n* » dans une section « =over » ... « =back ».
- Et, sans doute le plus important, utilisez des items cohérents entre eux : soit ce sont tous des « =item * » pour produire une liste à puces ; soit ils sont tous de la forme « =item 1 », « =item 2 », etc. pour produire une liste numérotée ; soit ils sont tous de la forme « =item truc », « =item bidule », etc. pour produire une liste de définitions. Si vous commencez par une puce ou par un numéro, continuez de même, puisque les traducteurs se basent sur le premier « =item » pour choisir le type de liste.

=cut

Pour terminer un bloc Pod, utilisez une ligne vide puis une ligne commençant par `=cut` puis encore une ligne vide. Ceci informe Perl (et les traducteurs Pod) que c'est à cet endroit que le code Perl recommence. (La ligne vide avant le `=cut` n'est pas techniquement indispensable mais beaucoup de vieux traducteurs Pod en ont besoin.)

=pod

La commande `=pod` en elle-même ne sert pas à grand chose si ce n'est de signaler à Perl (et aux traducteurs Pod) qu'une section Pod commence à cet endroit. Une section Pod peut commencer par *n'importe* quel paragraphe de commande. Une commande `=pod` ne sert donc qu'à indiquer une section Pod qui débute directement par un paragraphe ordinaire ou un paragraphe verbatim. Par exemple :

```
=item trucs()

Cette fonction fait des trucs.

=cut

sub trucs {
  ...
}

=pod

Souvenez-vous de vérifier son S<résultat :>

  trucs() || die "Ne peux pas faire des trucs !";

=cut
```

=begin *nomformat*

=end *nomformat*

=for *nomformat texte...*

Les commandes for, begin et end vous permettent d'utiliser des sections de texte/code/donnée qui ne seront pas interprétées comme du Pod normal mais qui pourront être utilisées directement par des traducteurs spécifiques ou qui pourront avoir un usage spécial. Seuls les traducteurs qui savent comment utiliser le format spécifié utiliseront cette section. Sinon elle sera complètement ignorée.

Une commande «=begin *nomformat*» puis quelques paragraphes et enfin une commande «=end *nomformat*» signifie que les paragraphes inclus sont réservés aux traducteurs comprenant le format spécial appelé *nomformat*. Par exemple :

```
=begin html

<hr> <img src="thang.png">
<p>Ceci est un paragraphe HTML</p>

=end html
```

La commande « =for *nomformat texte...* » indique que c'est uniquement ce paragraphe (le *texte* qui est juste après *nomformat*) qui est dans ce format spécial.

```
=for html <hr> <img src="thang.png">
<p>Ceci est un paragraphe HTML</p>
```

Les deux exemples ci-dessus produiront le même résultat.

La différence est qu'avec « =for », seul le paragraphe est concerné alors qu'entre le couple « =begin format » ... « =end format », vous pouvez placer autant de contenu que nécessaire. (Notez que les lignes vides après la commande =begin et avant la commande =end sont requises.)

Voici des exemples de l'utilisation de ceci :

```
=begin html

<br>Figure 1.<IMG SRC="figure1.png"><br>

=end html

=begin text

  ---------------
  |  foo        |
  |        bar  |
  ---------------

^^^^ Figure 1. ^^^^

=end text
```

Parmi les noms de format actuellement connus pour être reconnus par les formateurs, on trouve « roff », « man », « latex », « tex », « text » et « html ». (Des traducteurs Pod peuvent en considérer certains comme synonymes.)

Le nom de format « comment » est pratique pour placer des notes (juste pour vous) qui n'apparaîtront dans aucune version mise en forme de la documentation Pod :

```
=for comment
S'assurer que toutes les options sont documentées !
```

Quelques *nomformat*s utilisent le préfixe `:` (comme par exemple `=for :nomformat` ou `=begin :nomformat` ... `=end :nomformat`) pour indiquer que le texte ne doit pas être considéré comme brut mais qu'il contient en fait du texte Pod (c.-à-d. contenant éventuellement des codes de mise en forme) qui n'est pas à utiliser pour une mise forme normale (c.-à-d. qui n'est pas un paragraphe normal mais pourrait être utilisé, par exemple, comme note de bas de page).

`=encoding` *`codage`*

Cette commande permet de déclarer le codage utilisé dans le document. La plupart des utilisateurs n'en ont pas besoin ; mais si votre encodage n'est pas US-ASCII ou Latin-1 alors indiquez-le aux traducteurs Pod en plaçant une commande `=encoding` *`codage`* le plus tôt possible dans le document. Comme *codage*, utilisez l'un des noms reconnus par le module `Encode::Supported`. Exemples :

```
=encoding utf8

=encoding koi8-r

=encoding ShiftJIS

=encoding big5
```

N'oubliez pas, en utilisant une commande, que cette commande se termine à la fin de son *paragraphe*, pas à la fin de sa ligne. D'où dans les exemples ci-dessus, les lignes vides que vous pouvez voir après chaque commande pour terminer son paragraphe.

Quelques exemples de listes :

```
=over

=item *

Premier item

=item *

Second item

=back
```

```
=over

=item Foo()

Description de la fonction Foo

=item Bar()

Description de la fonction Bar

=back
```

2.4 Codes de mise en forme

Dans les paragraphes ordinaires et dans certains paragraphes de commande, plusieurs codes de mise en forme (appelés aussi « séquences internes») peuvent être utilisés :

`I<texte>` – texte en italique

Utilisé pour mettre en évidence (`faites I<attention !>`) et pour les paramètres (`redo I<LABEL>`).

`B<texte>` – texte en gras

Utilisé pour les options (`l'option B<-n> de perl`), pour les programmes (`certains systèmes proposent B<chfn> pour ça`), pour mettre en évidence (`faites B<attention !>`) et autres (`cette propriété s'appelle B<l'autovivification>`).

`C<code>` – du code

Présente le code dans une police type machine à écrire ou donne une indication que le texte est un programme (`C<gmtime($^T)>`) ou quelque chose liée à l'ordinateur (`C<drwxr-xr-x>`).

`L<nom>` – un hyperlien

Il y a différentes syntaxes, présentées ci-dessous. Dans ces syntaxes, `texte`, `nom` et `section` ne peuvent pas contenir les caractères '/' et '|' et les caractères '<' ou '>' doivent pouvoir s'associer.

- `L<nom>`
  Lien vers une page de documentation Perl (par exemple `L<Net::Ping>`). Notez que `nom` ne devrait pas contenir d'espaces. Cette syntaxe est aussi utilisé occasionnellement pour faire référence aux pages de manuel UNIX comme dans `L<crontab(5)>`.
- `L<nom/"section">` ou `L<nom/section>`
  Lien vers une section particulière d'une autre page de documentation. Par exemple `L<perlsyn/"Boucles for">`.
- `L</"section">` ou `L</section>` ou `L<"section">`
  Lien vers une section particulière de ce même document. Par exemple `L</"Méthodes objets">`.

Une section débute par un titre ou un item. Par exemple, `L<perlvar/$.>` ou `L<perlvar/"$.">` seront tous deux liés à la section qui débute par `=item $.` dans perlvar. Et `L<perlsyn/Boucles for>` ou `L<perlsyn/"Boucles for">` sont tous deux liés à la section débutant par `=head2 Boucle for"` dans perlsyn.

Pour contrôler le texte affiché comme lien, vous pouvez utiliser `L<texte|...>` comme dans :

- `L<texte|nom>`
  Lié ce texte à la page de documentation dont le nom est fourni. Par exemple `L<Messages d'erreurs de Perl|perldiag>`.
- `L<texte|nom/"section">` ou `L<texte|nom/section>`
  Lié ce texte à la section de la page de documentation dont le nom est fourni. Par exemple `L<postfix "if"|perlsyn/"Statement Modifiers">`
- `L<texte|/"section">` or `L<texte|/section>` ou `L<texte|"section">`
  Lié ce texte à la section de ce même document. Par exemple `L<les différents attributs|/"Données membres">`

Ou vous pouvez lier une page web :

- `L<scheme:...>`
  Lien vers un URL absolu. Par exemple `L<http://www.perl.org/>`. Mais notez que, pour différentes raisons, il n'existe pas de syntaxe du genre `L<text|scheme:...>`.

`E<entité>` – un caractère nommé

Très similaire aux « entités » HTML/XML `&foo;`.

- `E<lt>` – un < littéral (plus petit que)
- `E<gt>` – un > littéral (plus grand que)
- `E<verbar>` – un | littéral | (*bar*re *ver*ticale)
- `E<sol>` – un / littéral (barre oblique)

Les quatre codes ci-dessus sont optionnels sauf s'ils sont utilisés à l'intérieur d'un autre de code de mise en forme, en particulier `L<...>` ou lorsqu'ils sont directement précédés d'une lettre majuscule.

- `E<htmlentité>`
  Quelques entités HTML non numériques telle que `E<eacute>`, qui signifie la même chose que `&eacute;` en HTML – c.-à-d. un e minuscule avec un accent aigu.
- `E<nombre>`
  Le caractère ASCII/Latin-1/Unicode dont le code est le nombre. Le préfixe "0x" indique que *nombre* est en hexadécimal comme dans `E<0x201E>`. Le préfixe "0" indique que le nombre est en octal comme dans `E<075>`. Sinon, le *nombre* est considéré en décimal comme dans `E<181>`.
  Notez que les vieux traducteurs Pod peuvent ne pas reconnaitre l'octal et l'hexadécimal et que de nombreux traducteurs ne savent pas présenter correctement les caractères dont le code est supérieur à 255. (Certains traducteurs peuvent même choisir un compromis pour présenter les caractères Latin-1, en présentant un simple "e" à la place de `E<eacute>`.)

`F<nomfichier>` – utilisé pour un nom de fichier

Typiquement affiché en italique. Exemple : `F<.cshrc>`

`S<texte>` – texte contenant des espaces non sécables

Cela siginifie que les mots du *texte* ne doivent pas être séparés sur plusieurs lignes. Exemple : `S<$x ? $y : $z>`.

`X<nom de sujet>` – une point d'entrée d'index

Ce code est ignoré par la plupart des traducteurs mais certains peuvent l'utiliser pour construire un index. Il est toujours présenté comme la chaîne vide. Exemple : `X<URL absolu>`

`Z<>` – un code de mise en forme nul (sans effet)

C'est rarement utilisé. C'est l'un des moyens pour empêcher l'inteprétation d'un code E<...>. Par exemple, à la place de "`NE<lt>3`" (pour "N<3"), vous pourriez écrire "`NZ<><3`" (le code "Z<>" sépare le "N" et le "<" afin qu'ils ne soient pas considérés comme le début d'une (hypothétique) séquence "N<...>").

La plupart du temps, un simple couple inférieur/supérieur suffira pour délimiter le début et la fin de votre code de mise en forme. Mais il se peut que vous ayez besoin de placer un symbole supérieur (un signe « plus grand que », '>') dans un code de mise en forme. C'est très courant lorsqu'on souhaite présenter un extrait de code avec une police différente du reste du texte. Comme d'habitude en Perl, il y a plusieurs moyens pour le faire. Le premier consiste tout simplement à utiliser l'entité « plus grand que » via le code de mise en forme `E` :

```
C<$a E<lt>=E<gt> $b>
```

Ce qui produira : `$a <=> $b`.

Un moyen plus lisible et probablement plus "brut" est d'utiliser de délimiteurs qui n'imposent pas de codage spécial pour un simple ">". Les traducteurs Pod proposent cela en standard depuis perl 5.5.660 via les doubles délimiteurs ("<<" et ">>") qui peuvent être utilisés *si et seulement si il y a un espace après le délimiteur ouvrant et un espace avant le délimiteur fermant !*. Par exemple :

```
C<< $a <=> $b >>
```

En fait, vous pouvez utiliser le nombre d'inférieurs et de supérieurs que vous souhaitez tant qu'il y en a autant dans le délimiteur ouvrant que dans le délimiteur fermant et si vous vous assurez qu'un espace suit immédiatement le dernier « < » du délimiteur ouvrant et qu'un autre espace précède immédiatement le premier « > » du délimiteur fermant (ces espaces seront ignorés). Les exemples suivants fonctionneront donc aussi :

```
C<<< $a <=> $b >>>
C<<<< $a <=> $b >>>>
```

Et ils signifient exactement la même chose que :

```
C<$a E<lt>=E<gt> $b>
```

Prenons un autre exemple : supposons que vous voulez présenter l'extrait de code suivant dans un style `C<>` (code) :

```
open(X, ">>thing.dat") || die $!
$foo->bar();
```

Vous pourrez le faire comme cela :

```
C<<< open(X, ">>thing.dat") || die $! >>>
C<< $foo->bar(); >>
```

qui est certainement plus facilement lisible que l'ancien méthode :

```
C<open(X, "E<gt>E<gt>thing.dat") || die $!>
C<$foo-E<gt>bar();>
```

Tout cela est actuellement accepté par pod2text (Pod::Text), pod2man (Pod::Man) et tout autre traducteur pod2xxx et Pod::Xxxx qui utilise Pod::Parser version 1.093 ou supérieure.

2.5 L'objectif

L'objectif est la simplicité d'utilisation, pas la puissance expressive. Les paragraphes ont l'air de paragraphes (des blocs) pour qu'ils ressortent visuellement, et on peut les faire passer facilement à travers `fmt` pour les reformater (c'est F7 dans ma version de **vi** ou Esc-Q dans ma version de **emacs**). Je voulais que le traducteur laisse les `'`, les `` ` `` et les `"` tranquilles en mode verbatim pour que je puisse copier/coller ces paragraphes dans un programme qui marche, les décaler de 4 espaces, et l'imprimer, euh, mot pour mot. Et probablement dans une fonte à chasse fixe.

Le format Pod est certainement insuffisant pour rédiger un livre. Pod essaie juste d'être un format infaillible qui puisse servir de source pour nroff, HTML, TeX et autres langages de balises, lorsqu'ils sont utilisés pour de la documentation en ligne. Des traducteurs existent pour **pod2text**, **pod2html**, **pod2man** (c'est pour nroff(1) et troff(1)), **pod2latex** et **pod2fm**. D'autres encore sont disponibles sur CPAN.

2.6 Incorporer du Pod dans les modules Perl

Vous pouvez inclure de la documentation Pod dans vos scripts et vos modules Perl. Commencez votre documentation par une ligne vide puis une commande « =head1 » et terminez-la par une commande « =cut » et une ligne vide. Perl ignorera le texte en Pod. Regardez n'importe lequel des modules fournis en standard pour vous servir d'exemple. Si vous souhaitez mettre votre Pod à la fin du fichier, et si vous utilisez un __END__ ou un __DATA__ comme marque de fin, assurez-vous de mettre une ligne vide avant votre première commande Pod.

```
__END__

=head1 NAME

Time::Local - efficiently compute time from local and GMT time
```

Sans cette ligne vide avant `=head1`, de nombreux traducteurs ne reconnaitront pas `=head1` comme le début d'une section Pod.

2.7 Conseils pour écrire en Pod

- La commande **podchecker** permet de vérifier le respect de la syntaxe Pod. Par exemple, elle vérifie les lignes entièrement blanches dans les sections Pod ou les commandes et les codes de mise en forme inconnus. Vous pouvez aussi passer votre document au travers d'un ou plusieurs traducteurs Pod et vérifier le résultat (en l'imprimant si besoin est). Certains problèmes rencontrés peuvent être liés à des bogues des traducteurs. À vous de décider si vous voulez les contourner ou non.
- Si vous êtes plus à votre aise en rédigeant du HTML que du Pod, vous pouvez rédiger votre documentation en HTML simple puis la convertir en Pod grâce au module expérimental Pod::HTML2Pod (disponible sur CPAN) et enfin vérifier le code obtenu. Le module expérimental Pod::PXML peut aussi être utile.
- De nombreux traducteurs Pod ont absolument besoin d'une ligne blanche avant et après chaque commande Pod (commande =cut y compris). Quelque chose comme ça :

```
# - - - - - - - - - - - -
=item $firecracker->boom()
```

```
This noisily detonates the firecracker object.
=cut
sub boom {
...
```

...amènera ces traducteurs Pod à ignorer totalement la section Pod.

À la place, préférez quelque chose comme ceci :

```
# - - - - - - - - - - - -
=item $firecracker->boom()
This noisily detonates the firecracker object.
=cut
sub boom {
...
```

– Certains traducteurs Pod anciens ont absolument besoin de paragraphes (incluant les paragraphes de commande comme "=head2 Fonctions") séparés par des lignes *complètement* vides. Si vous avez des lignes apparament vides mais contenant en fait des espaces, elles ne seront pas reconnues comme séparateurs par ces traducteurs et provoqueront peut-être de mauvaises mises en forme.
– Des traducteurs Pod anciens ajoutent quelques mots autour de certains liens L<> de telle manière que `L<Foo::Bar>` deviendra par exemple "the Foo::Bar manpage". Vous ne pouvez donc pas écrire des choses telles que `la documentation L<bidule>` si vous voulez que le résultat reste compréhensible. À la place, écrivez `la documentation L<bidule|bidule>` ou `L<la documentation bidule|bidule>` pour contrôler l'apparence du lien.
– Un texte qui dépasse la 70e colonne dans un bloc verbatim peut être arbitrairement coupé par certains traducteurs.

3 VOIR AUSSI

perlpodspec, POD: documentation intégrée in *perlsyn*, *perlnewmod*, *perldoc*, *pod2html*, *pod2man*, *podchecker*.

4 AUTEUR

Larry Wall, Sean M. Burke

5 TRADUCTION

5.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.10.0. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

5.2 Traducteur

Traduction initiale : Roland Trique <*roland.trique@free.fr*>. Mise à jour : Paul Gaborit <paul.gaborit at enstimac.fr>.

5.3 Relecture

Gérard Delafond

6 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.